PATENT ABSTRACTS OF JAPAN

(11)Publication number : 64-070838
(43)Date of publication of application : 16.03.1989

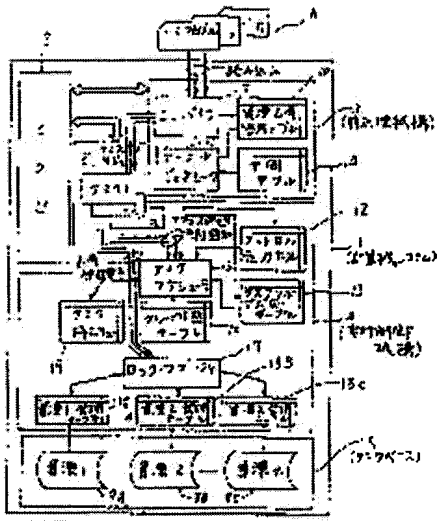
(51)Int.Cl. G06F 11/28
G06F 9/44
G06F 9/46

(21)Application number : 62-227799
(22)Date of filing : 11.09.1987

(71)Applicant : HITACHI LTD
(72)Inventor : HIROTA ATSUHIKO
OWAKI TAKASHI
OSHIMA KEIJI

(54) EXCLUSIVE CONTROL SYSTEM

(57)Abstract:
PURPOSE: To prevent the generation of deadlock by extracting the using order of resources by a program compiler for plural programs using plural resources in common and previously extracting the existence of generation of deadlock on the basis of the extracted information.
CONSTITUTION: The access order of resources 9 in a data base 5 is extracted by the compiler 7 and stored in a resource occupation table 10. At the time of ending the compilation of all source programs, a deadlock checking table 12 and a table 13 corresponding to task groups to be used for an execution control mechanism 14 are formed by means of a table generator 11. At the time of operating a task group 8, the mechanism 4 receives information for occupying/releasing resources and schedules tasks so that no deadlock is generated between the tasks on the basis of information stored in the tables 12, 13 and a group state table 16.



⑲日本国特許庁(JP)　⑪特許出願公開

⑫公開特許公報(A)　昭64-70838

⑤1Int.Cl.⁴	識別記号	庁内整理番号	⑬公開　昭和64年(1989)3月16日
G 06 F 11/28	340	A-7343-5B	
9/44	320	E-8724-5B	
9/46	340	G-7056-5B	審査請求　未請求　発明の数 1　(全11頁)

⑤4発明の名称　排他制御方式

⑳特　願　昭62-227799
㉒出　願　昭62(1987)9月11日

⑦2発明者　廣田　敦彦　茨城県日立市大みか町5丁目2番1号　株式会社日立製作所大みか工場内
⑦2発明者　大脇　隆志　茨城県日立市大みか町5丁目2番1号　株式会社日立製作所大みか工場内
⑦2発明者　大島　啓二　茨城県日立市大みか町5丁目2番1号　株式会社日立製作所大みか工場内
⑦1出願人　株式会社日立製作所　東京都千代田区神田駿河台4丁目6番地
⑦4代理人　弁理士　鵜沼　辰之　外1名

明　細　書

1．発明の名称

排他制御方式

2．特許請求の範囲

1．並列に動作する複数のプログラムにより複数の資源を共用するに際し、該プログラム毎の該資源に対する使用順序を定義し、この定義された情報に基づき該プログラム間のデッドロック発生の可能性の有無を該プログラムの実行以前に抽出し、この抽出された情報に基づきデッドロックの発生を抑止する排他制御方式において、複数の前記資源を共用する複数の前記プログラムの資源操作記述により前記プログラムをコンパイルするコンパイラによって各前記プログラムから各前記資源の使用順序を抽出することを特徴とする排他制御方式。

2．前記プログラムの資源操作記述を、資源操作範囲がブロック構造となる構造化操作記述言語を用い、該構造化操作記述言語のコンパイラにより前記プログラムのコンパイル時に該プログラムに記述されている前記資源操作範囲のブロックネストの関係から前記資源の使用順序を該プログラム毎に抽出することを特徴とする特許請求の範囲第1項記載の方式。

3．前記コンパイラによって抽出した各前記資源の使用順序，該使用順序により決定した前記プログラムの同時実行可能性の情報を表示することを特徴とする特許請求の範囲第1項または第2項記載の方式。

3．発明の詳細な説明

〔産業上の利用分野〕

一般に、電子計算機システム上の実現されるデータベースシステムにおいて、複数のプログラムがデータベース中の資源を共通に利用し、非同期にデータベースをアクセスする。このため、同一の資源あるいは資源群に対する複数プログラムからのアクセス、例えば更新動作と参照動作、あるいは更新動作と更新動作、が同時に発生しないようにして、データベース中の資源の不整合や、他のプログラムが更新途上である資源の読み出しを

防止する必要がある。

本発明は、このようなデータベース中の資源の排他制御に係り、特に各プログラムに資源の排他制御のための手続きなどを負担させることなく、効率のよい、かつ信頼度の高いデータの排他制御を実現するために好適な、データベースシステムの資源の排他制御方式に関するものである。

〔従来の技術〕

前述のようなデータベース中のデータの排他制御を行うに当つては、一般に、次に示すようなデータのロッキング方式がとられているロッキング方式とは、プログラムがある一連のデータベース処理を行う際、アクセスしようとするデータ群を前もつて占有（ロック）し、アクセスが終了した時点で、占有していたデータ群を解放（アンロック）する、といつた手続を各プログラムが実施し、占有しようとしたデータ群が、他のプログラムによつて占有された場合には、その占有状態が解放されるまで待つ、といつた規約をもつことにより、あるプログラムが処理中のデータを他のプログラムが同時に使用することがないようにする方式である。

しかしながら、ロッキング方式には、各プログラムが無秩序にデータの占有を行うと、プログラム間で互いに他のプログラムが占有しているデータが解放されるのを待ち合い、永遠にプログラムが動けなくなつてしまう状態、すなわちデッドロックが発生することがあるという問題が知られている。

このデッドロックの問題に対処するために、以下のいずれかの方法がとられている。

イ）データを占有する際に、デッドロックが発生しないような規制を設ける。

ロ）デッドロックが発生したことを検出し、デッドロック状態の回復を行う。

以下、具体的な従来技術について説明する（参考文献として、①共有データベースの諸問題に対する理論（上林）情報処理Ｖｏｌ．２４，Ｎｏ８（１９８３－３）、②データベース排他制御方法（特願昭60－73817）がある）。

(1) 一括ロック方式

アクセスしようとするすべてのデータを一括して占有し、アクセスが終了したら、占有したデータを解放する方法である。

すなわち、この方式は、占有の回数をただ一度だけに限定することによつて、デッドロックの発生可能性を完全になくしたことが特長である。

(2) 木規約に従うロック方式

すべてのプログラムが、データを逐次占有していく際の占有順序を、同一データが複数回出現することのない、ある特定の半順序集合に統一する方法である。この半順序集合で示されたロック規約を木規約という。

すなわち、この方式は、すべてのプログラムのデータの占有順序を統一化することによつて、デッドロックの発生可能性を完全になくしたことが特長である（参考文献①，３．１節参照）。さらにプログラムごとのデータ占有順序を予め定義させ、この規約を用いて、プログラムがデッドロックを起こさないように、実行時にスケジューリングを行う方式もとられている（参考文献②参照）。

(3) デッドロック検出方式（ロールバック方式）

各プログラムからは自由に必要となるデータを占有させることとし、それに伴つて発生するデッドロックを検出し、検出した場合にプログラムの処理結果を無効とし（ロールバックし）、デッドロック状態からの回復を行うとともに、無効化されたプログラムを一定時間遅延されたうえで、再実行させるといつた方式である（参考文献①，２．３節参照）。

〔発明が解決しようとする問題点〕

上記従来技術には、それぞれ次のような問題があつた。

(1) 一括ロック方式

この方式の欠点は、プログラムがある意味ある処理単位内で必要とするすべてのデータに対して、そのうちの少なくとも１つのデータをアクセスしようとした時点で、一括してデータを

占有しなくてはばらないため、共有データ部分をもつ複数のプログラム同士のデータ・アクセスは、すべてシリアルに実行されることになる。このことは、プログラムの応答性に悪影響を及ぼす。

また、プログラムの記述性の面からも、占有記述の際に、占有するデータを予め調べて列記しなくてはならないため、プログラム作成者に排他制御に対する意識を強いることとなり、望ましいものではない。

(2) 木規約に従うロック方式

一括ロック方式に比較し、データをアクセスする時点にて、個々にデータを占有していくため、プログラムの応答性の面では改善されている。

しかし、プログラムをある決められた木規約に従つて作成せねばならず、プログラムからのデータ・アクセスの自由度を大幅に制約することににになり、結果として、プログラム作成者に排他制御に対する意識を強いることになり、望ましいものではない。

また、この規約は、すべてのプログラムに最適となるように決定すべきであるが、それを決めることが、データベース管理者の非常な負担となる。

以上述べたいずれの方式を用いても、データの占有・解放といつた特別の手続きを、プログラム作成者に強要することとなり、プログラムの生産性を低下させるばかりでなく、誤使用によるシステムの信頼性・応答性の低下をひきおこす危険性を伴つている。

(3) デツドロック検出（ロールバック方式）

本方式は、前記の2つの方式と比較すると、プログラマやデータベース管理者に対する負担は少ない。

しかしながら、デツドロック検出，ロールバック，再実行を行うに当り、以下に示すプログラムの性能面での問題が生じるという欠点がある。

(a) デツドロック検出オーバヘツド

システムは、デツドロック検出のための監視を行う必要があり、また検出した際に、無効化するプログラムの特定が必要となり、システムの負荷が増大する。

(b) ロールバック準備オーバヘツド

ロールバックの際には、更新されたデータを、更新前の状態へ回復する必要があるが、そのためには、ログの取得といつた前準備が必要となり、これは、プログラムの応答性低下の要因となる。

(c) ロールバック実施オーバヘツド

プログラムの更新したデータをすべてもとの状態へ戻さねばならず、相当の処理時間を要する。

さらに、無効化したプログラムの更新したデータが、デツドロック発生時点ですでに解放されており、他のプログラムがそのデータを用いて処理を行つていたりする場合には、他のプログラムにまで無効化、ロールバックが波及する場合もある(ロールバックの連鎖)。ロールバック連鎖を防ぐ方法も考察されてはいるものの、いずれにせよ本方式を用いている限りでは、ロールバック実施に相当な処理時間を要する（参考文献①、2．4節参照）。

(d) 再試行オーバヘツド

無効化されたプログラムは、再試行する必要があり、これも相当な処理時間を要する。

上記オーバヘツドを勘案すると、プログラムの実質的な並行処理効率は相当低下するものと考えられ、またプログラムの応答性もデツドロック発生有無により大きく変化してしまうため、リアルタイム処理のような高い応答性の要求される分野への適用は不可能である。

また、上記文献②（特願昭61－233849）の場合、システム管理者が予めデータの占有順序を定義しなければならず、システム管理者の負担が大であり、かつ誤りの発生する可能性もあつた。

本発明の目的は、以上述べた従来技術の問題点である、

(1) 共用データの占有・解放といつた特別な手続

きや、データアクセス順序に対する制約を、データベース利用者へ強要すること、

(2) デッドロックの発生や、ロールバックといったプログラムの実質的な処理効率の低下、

を解決することにより、プログラムの生産性、信頼性を向上させるデータの排他制御を実現することにある。

〔問題点を解決するための手段〕

上記問題点は、並列に動作する複数のプログラムにより複数の資源を共用するに際し、該プログラム毎の該資源に対する使用順序を定義し、この定義された情報に基づき該プログラム間のデッドロック発生の可能性の有無を該プログラムの実行以前に抽出し、この抽出された情報に基づきデッドロックの発生を抑制する排他制御方式において、複数の前記資源を共用する複数の前記プログラムの資源操作記述により前記プログラムをコンパイルするコンパイラによって各前記プログラムから各前記資源の使用順序を抽出する排他制御方式によって解決される。

〔作用〕

複数の資源を共用する複数のプログラムの資源操作記述により前記プログラムをコンパイルするコンパイラによって各前記プログラムから各前記資源の使用順序を抽出し、この抽出した情報に基づき該プログラム間のデッドロック発生の有無を該プログラム実行以前に抽出し、この抽出した情報に基づきデッドロックの発生を抑制するよう制御する。

〔実施例〕

以下、本発明による一実施例について詳述する。

第１図は、本発明を適用する計算機システムの概略構成図である。計算機システム１は、ＣＰＵ２，前処理機構３，実行制御機構４および記憶装置上に実現されるデータベース５からの構成されている。

前処理機構３はユーザが作成し、本計算機システム上でタスクとして動作するプログラムのソースプログラム群６を読み込み、コンパイラ７を用いてこれを計算機上で動作可能な形式に翻訳し、タスク８として生成するとともに、コンパイラ７にてソースプログラムからそのプログラムのデータベース５中の資源９に対するアクセス順序（占有順序）を抽出し、資源占有順序テーブル１０へ記憶する。すべてのソースプログラムのコンパイルが終了した時点で、テーブルジェネレータ１１を用いて、実行制御機構４が実行制御のために用いるデッドロックチェックテーブル１２およびタスクグループ対応テーブル１３を生成する。このとき、テーブルジェネレータ１１は、テーブル生成のための一時的情報格納のため、中間テーブル１４を用いる。タスクグループ対応テーブル１３は、同じような占有順序をもつタスク群をグルーピングし、そのグループ識別子（ＧＩＤ）と、タスクごとに付与されたタスク識別子（ＴＩＤ）との対応を表わすテーブルである。デッドロックチェックテーブル１２は、ＧＩＤごとに他のグループのタスクとのデッドロック発生可能性有無を記憶しているテーブルである。

実行制御機構４は、タスク群８が動作する際、資源を占有・解放する通知を受け取り、受け取った情報と、デッドロックチェックテーブル１２，タスクグループ対応テーブル１３およびグループ状態テーブル１６に格納されている情報に基づきタスク相互間でデッドロックが発生しないようタスクの動作をスケジューリングするタスクスケジューラ１５と、タスク群８からの個々の資源９に対する占有・解放要求に基づき、資源ごとに作成された資源管理テーブル１８を用いて、占有・解放を実施するロックマネージャ１７から成る。グループ状態テーブル１６は、各グループに属しているタスクのうち、スケジューリング可能（資源占有可能）状態となっているタスク数をグループごとに格納している。また、タスクスケジューラ１５は、スケジューリング待ちとなっているタスクの待ち管理のために、タスク待ちキュー１９を使用する。資源管理テーブル１８は、現在資源を占有しているタスクのＴＩＤを格納しているテーブルである。

なお、本実施例では、単一計算機システムの例

となつているが、例えば前処理機構と、実行制御機構が別々のＣＰＵにより動作し、ＣＰＵ間に適当な交信機能を持たせたような複合計算機システムであつても、本発明に何ら影響を与えない。

第２図は、本計算機システム上でタスクとして動作するプログラムの動作内容を示したソースプログラムの一例である。ソースプログラムは、例えばPASCALやＣ言語等の「構造化プログラミング言語」を親言語とし、その中に資源に対する操作記述範囲を入口、出口が１箇所となるブロック構造で規定した「構造化されたデータベース操作言語」が埋め込まれた形となつていることが特徴である。本例では、処理の入口を「ＷＩＴＨ　資源名　ＤＯ　ＢＥＧＩＮ」、処理の出口を「ＥＮＤ;」というキーワードを用いたブロック構造で規定している。

なお、ブロック構造の規定の仕方（キーワード等）については、親言語となるプログラミング言語と区別可能な、例えば言語処理系における字句解析，構文解析等においてあいまいさが発生しないような記法であれば、いかなるものであつてもかまわない。また、データベース操作言語におけるブロック構造は、親言語におけるブロック構造，例えばPASCAL 言語におけるBEGIN～ＥＮＤ；と置き換え可能な位置に記述されていなくてはならない。したがつて、データベース操作言語のブロック構造は、親言語におけるブロック構造とネスト（入れ子）の関係になるか、あるいは連接または並置の関係になるかのいずれかとなる。また、データベース操作言語自身のブロック構造のブロック間の関係においても同様である。また、記述する資源名称のネストの上位，下位の関係についても特に制約はもたないものとする。本記述例では、ブロック構造を規定する部分以外の記述は省略して記した。

なお、図中の操作記述範囲は、そのブロックの内側にあるブロック内をも含むものとする。

上記規則に従つていない場合は、ソースプログラムのコンパイルの際チエックアウトされ、ユーザによりソースプログラムの修正が実施される。

以下、第１図に示す計算機システムの各構成要素の動作の詳細を説明する。

コンパイラ７は、第２図に示したような形式で記述されたソースプログラムを読み込み、通常のプログラミング言語のコンパイラと同様、ソースプログラムを解析し、実行可能な形式に変換する。

この際、データベース操作言語のブロック構造の最も外側の入口、出口へ、タスクスケジューラ１５に対する資源アクセス開始，終了通知用システムコールを、またすべてのブロック構造に対応する入口，出口に、ブロック単位に指定された資源に対する占有，解放を、ロックマネージャ１７に要求するためのシステムコールを埋め込む。このとき、すでに外側のブロックにて出現している資源名に対応する資源と同一の資源が内側のブロックに現われた場合は、その内側のブロックの入口、出口への該システムコールの埋め込みは行わないものとする。すなわち、１つの資源に対する２重占有は行わないようにする。

また、一方、そのプログラム内にあるデータベース操作言語のブロック構造のネスト関係から、資源の占有順序情報を抽出する。

資源占有順序は、ブロックに対応する資源名を節点，外側のブロックを始点，内側のブロックを終点とする有向枝をもつ有向グラフの形となる。

第３図に、ソースプログラムから、上記の有向グラフを抽出する処理例の概略フローを示す。

本処理フローのキーワードは、第２図に示したソースプログラム記述に準じている。また、本プログラムは、有向グラフを計算機上で表現しやすいように、有効枝に対応した（終点，始点）の２つ組の集合を、結果として出力するようにしている。

第４図に、第２図のプログラムをソースプログラムとし、第３図の処理フローを適用した際の結果を(a) に示し対応する有向グラフを(b) に示す。

コンパイラ７は、各スタックごとにこの有向グラフを作成し、資源占有順序テーブル１０へ格納する。なお、通常の場合、１つのタスクは１本のメインプログラムと複数のサブプログラムからなり、

それぞれのコンパイル単位は別となる。このような場合は、コンパイル単位ごとに有向グラフを作つておくとともに、プロツク内で使用しているサブルーチン呼び出しを抽出しておき、その関係からそのプログラムがリンクエディツトされるのと同様、有向グラフもリングエディツト（マージ）することにより、最終的には、タスクごとの有向グラフが抽出できるようになる。

第５図は、資源占有順序テーブル１０の実現例(a)と対応する有向グラフ(b)を示す。ここでは、第４図で示した結果に、ＴＩＤが付与された３つの組の形式で記憶している。また、２重枝（第４図の例におけるＢからＡへ向う枝）は、１つの枝として記憶しているとともに、推移枝（第４図の例におけるＣ→Ａの枝）は、他の枝（Ｃ→Ｂ，Ｂ→Ａ）から推移可能であるので、省略して記憶する。なお、第２図のプログラムは、ＴＩＤ＝１の場合として表わされている。

テーブルジエネレータ１１は、本計算機システム上でデータベースのアクセスを実施するすべてのタスクについて資源占有順序テーブル１０に占有順序情報が登録された時点で起動され、まずタスク相互間でデツドロツク発生可能性があるかどうかを示す中間テーブル１４を生成する。第６図に、第５図に対応して作成される中間テーブルの例を示す。

なお、第６図において、○は同時動作してもデツドロツクが発生しないことを、×はデツドロツクの発生する可能性のあることを示す。なお、計算機システム上では、例えば○を０、×を１などし、記憶する。

第７図は、第６図に示すような中間テーブルの１つの要素の○，×を伴定するための処理例の概略フローである。デツドロツク発生可能性の有無は、基本的には、資源占有順序を示す有向グラフを、対象とするタスク同士間でマージしたとき、そこにグラフをマージしたことにより強連結成分が存在するか否かを判定することにより求められる。

有向グラフの強連結成分は、グラフ上の任意の節点をとつたときに、そこから出発して有向枝をたどり、再び出発点へ戻つてこれるような径路が１つでも存在すれば、そのグラフ上に存在する。なお、強連結成分の存在有無は、推移枝の存在有無により影響を受けない。強連結成分の存在有無は、例えば次のような処理で求めることができる。

(1) 与えられた有向グラフに、そのグラフを構成する枝から推移されるすべての有向枝を付加する。

(2) 付加された推移枝から推移される推移枝も、同様に再帰的に付加する。

(3) 得られた有向グラフのすべての枝の組み合わせの中に、１組でも始点と終点の関係が逆の枝が存在すれば、強連結成分が存在する。

なお、この中間テーブル１４は、上三角形分と下三角成分が対称であるため、実際にはそのどちらか一方を求めることにより作成可能である。

なお、１つもネストしないタスクのＴＩＤは、資源占有順序テーブルに出現しないが、このタスクに対応する各行，各列はすべて○とする。

テーブルジエネレータは、次にタスクのグルーピングを実施する。第８図は、グルーピングの一処理例の概略フローである。

第８図の処理は、実行時のオーバヘツドを減少するために、デツドロツク発生可能性のないタスク同士で、他タスクとのデツドロツク発生可能性が同一であるタスクをグルーピングするものである。グルーピングした結果は、デツドロツクチエツクテーブル１２およびタスクグループ対応テーブル１３へ格納される。

第９図は、第６図の中間テーブル１４をグルーピングした結果であるデツドロツクチエツクテーブル１２(a)と、タスクグループ対応テーブル１３(b)の例である。

また、前処理装置３は、例えばデツドロツクチエツクテーブル１２，タスクグループ対応テーブル１３の内容、および資源占有順序テーブル１０の内容を、例えばＣＲＴやプリンタといつた出力装置にプログラムの同時実行可能性をユーザが知るための情報として出力する。このことにより、

ユーザは、プログラムを実際に動作させることなしにタスクの並行処理性を見積ることが可能となり、また並行処理効率を悪化させる要因となる。アクセス順序の異なるタスクを摘出し、プログラム修正を行うことも可能となる。

前処理機構３は、以上示した処理を実際にタスクが動作する前に行うため、タスクの実行におけるオーバヘッドの増大は伴わない。

以下、タスク実行時における排他制御実施方法の一実施例につき詳述する。

第１０図は、実行制御機構４におけるタスクスケジューラ１５の処理例の概略フローである。

タスク８は、動作開始後、コンパイラ７によつて埋め込まれた資源アクセス開始通知用システムコールにて、タスクスケジューラ１５に対して資源９のアクセス開始を通報する。タスクスケジューラ１５は、まずタスクグループ対応テーブル１３から、通報のあつたタスク８が所属しているグループのＧＩＤを取り出し、次にグループ状態テーブル１６を参照し、同一グループに属しているタスク８が動作中がどうかを伴定する。グループ状態テーブル１６は、グループごとにスケジューリング（動作）中のタスク８数を格納している。もし、動作中（タスクカウンタ≠０）であるならば、当該タスク８もスケジューリング可能状態となるため、当該タスク８の所属するグループに対応するタスク８数カウンタをカウントアツプする。

もし、当該タスクの所属グループのタスクが動作中でない（タスクカウンタ＝０）ならば、デツドロツクチエツクテーブル１２から、デツドロツクの発生可能性のあるグループのＧＩＤを取り出す。もし存在していなければ、前記同様タスクカウンタをカウントアツプし、スケジューリング状態となる。存在している場合は、そのＧＩＤを用いグループ状態テーブル１６のタスクカウンタを参照し、１つでも動作中の場合は、タスク待ちキュー１９に当該タスク８のＴＩＤを登録し、待ち状態となる。１つも動作中でない場合には、グループ状態テーブル１６を参照し、動作中グループ数（タスクカウンタが１以上となつているグループの数）が２つ以上あれば、タスク待ちキュー１９に当該タスクのＴＩＤを登録し、待ち状態となる。グループ数が１つの場合は、グループ状態テーブル１６の当該タスクの所属ＧＩＤのタスクカウンタをカウントアツプする。

一方、タスク８は、資源に対する一連のアクセスを完了すると、資源アクセス終了通知用システムコールにて、タスクスケジューラ１５に対して資源アクセスの完了を通報する。タスクスケジューラ１５は、通報のあつたタスク８のＴＩＤを用いて、タスクグループ対応テーブル１４から当該タスク８の所属ＧＩＤを取り出し、グループ状態テーブル１６のＧＩＤに対応するタスクカウンタをカウントダウンする。さらに、アクセス完了に伴つて、待ちとなつているタスク群が実行可能となる場合があるため、タスク待ちキュー１９に登録され、待ちとなつているタスク８について、要求時と同一の処理（第１０図参照）を実施し、再びデツドロツクチエツクを行い、デツドロツクが発生しない場合は、そのタスク８を動作させる（スケジューリング可能性態とする）。

以上説明したように、タスクスケジューラ１５は、デツドロツクを発生させる可能性のあるタスク８を同時に実行させないように制御するため、タスク８間のデツドロツク発生を完全に回避できる。

一方、起動され実行中のタスク８は、個々の資源の使用開始時点において、資源ロツクのシステムコールを発行することにより、ロツクマネージャ１７に対し、タスク８のＴＩＤと使用対象となる資源の識別子を渡たし、資源ロツク要求を行う。

これに対して、ロツクマネジヤ１７は、資源識別子に対応する資源が他のタスク８によりロツク中か否かを調べめるために、資源管理テーブル１８Ａ，１８Ｂ，１８Ｃのうちの該当テーブルを参照する。資源管理テーブル１８Ａ，１８Ｂ，１８Ｃは、個々の資源がロツク中か否かの情報とロツクタスク８のＴＩＤ（ロツク中の場合のみ）の格納している。ロツクマネージヤ１７は、参照結果よりロツク要求を出したタスク以外のタスク

8がロック中の場合、ロック要求を出したタスク8の資源がアンロックされるまで待ちとする。ロック中でない場合は、該当資源管理テーブル18中の対象資源のロック情報をロック中であると更新し、ロック要求を出したタスク8のロックを認可する。

一方、タスク8は、個々の資源の使用終了時点において、資源アンロックのシステムコールを発行することにより、ロックマネージャ17に対しタスク8のTIDと使用終了する資源の識別子を渡し、資源アンロック要求を行う。これに対し、ロックマネージャ17は、資源管理テーブル18中の該当する資源のロック状態情報をアンロック状態とし、資源のアンロック待ちとなっているタスク8を待ち状態から実行状態へ回復する。このことにより、1つの資源に対する複数タスク8からの同時操作を排除し、データベース中のデータの完全性破壊防止できる。

以上説明したように、実行制御機構として、タスクグループ対応テーブル13，デッドロックチェックテーブル12，グループ状態テーブル16およびタスクスケジューラ15を具備し、デッドロックの発生可能性のあるタスクを待ちにすることにより、デッドロック発生を完全に回避し、またデータの完全性の破壊を防止した効率のよい排他制御が実現できる。

なお、本実施例においては、2グループ間のデッドロックチェックしか行っていないため、2グループしか同時にスケジューリングできないが、同様の方法により、nグループ間までのチェックを行うことにより、nグループ同時にスケジューリングできるようになることはいうまでもない。

また、ブロックネストのないタスク（対応する有向グラフ上に節点のみで枝が1つも存在しないタスク）については、デッドロック要因とはならないため、タスクスケジューラ15による管理対象外として、常にスケジューリングできるようになることもいうまでもない。

また、本実施では、資源に対する占有をタスク8間で排他的に行う排他ロック方式につき説明したが、検索のみ行う場合においては、同時に資源を占有できる共用ロック方式を可能とした場合においても、コンパイラによりデータのアクセス種類（更新を行うか否か）をも抽出し、それに基づき、ロックモード（排他共用）を決定し、デッドロック発生可能性のあるタスク8を決める際に、双方とも共用ロックモードであれば、待ちが発生しないことをデッドロックチェックの判定条件に付加することなどにより、より並行性の高い排他制御が実現できることも容易に類推可能である。

以上説明したように、本実施例のデータの排他制御方式を用いると、

(1) プログラマ，データベース管理者ともに、排他制御に対する事項（手続きの記述やデータ操作順序の制約）を一切意識することなく、データベース操作が行える。

(2) デッドロックを完全に回避した並行処理動率の高い排他制御が実現でき、また実行時のオーバーヘッドも小さい。このことにより、オンライン・リアルタイム分野の適用も可能である。

(3) ユーザが介在しないため、高い信頼性を実現できる。

といつた効果があり、プログラムの生産性，信頼性の向上が図れる。

〔発明の効果〕

本発明によれば、複数の資源を共用する複数のプログラムの資源操作記述により前記プログラムをコンパイルするコンパイラによつて各前記プログラムから各前記資源の使用順序を抽出できるので、プログラム間のデッドロックの発生の有無を該プログラムの実行以前に抽出してデッドロックの発生を防止できるという優れた効果がある。

4．図面の簡単な説明

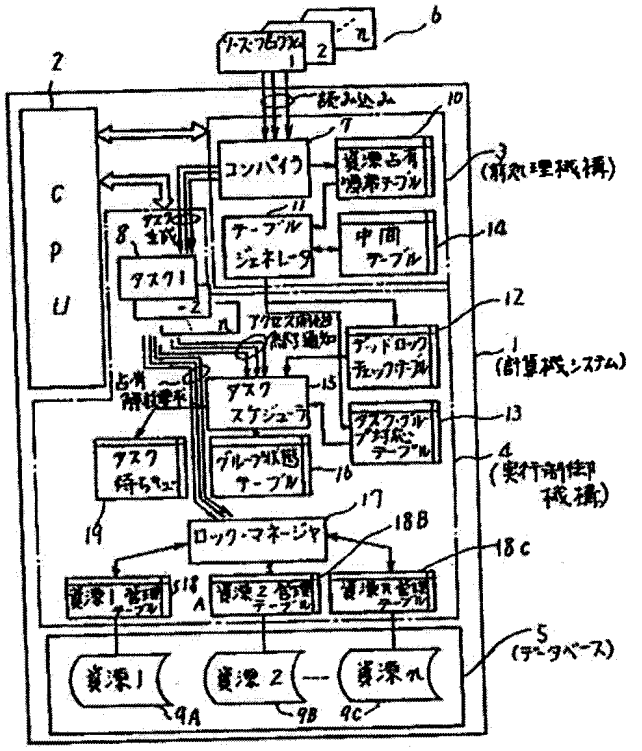
第1図は本発明の一実施例の計算機システムの概略構成図、第2図は同計算機システムのプログラム記述例、第3図はコンパイラの概略処理フロー、第4図(a)，(b)はコンパイラの出力例を対応グラフ、第5図(a)，(b)は資源占有順序テーブルの内容例と対応グラフ、第6図は中間テーブルの例を示す図、第7図はテーブルジェネレータ

の概略処理フロー、第８図はテーブルジネレータのグルーピング処理フロー、第９図(a)，(b)はデットロックチエックおよびタスクグループ対応テーブル内容例、第１０図はタスクスケジューラの概略処理フローである。

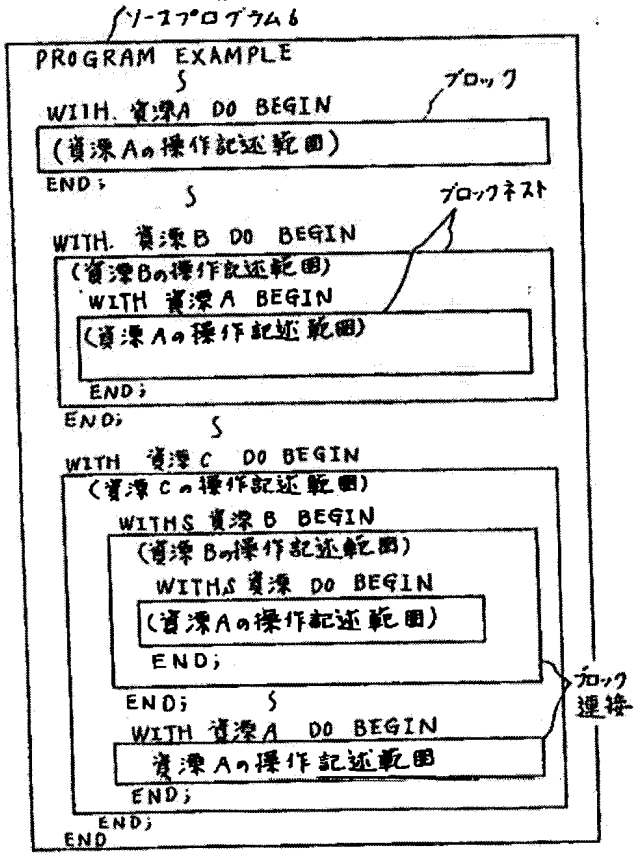
１…計算機システム、２…ＣＰＵ、３…前処理機構、４…実行制御機構、５…データベース、６…ソースプログラム、７…コンパイラ、８…タスク、９…資源。

代理人　弁理士　鵜沼辰之

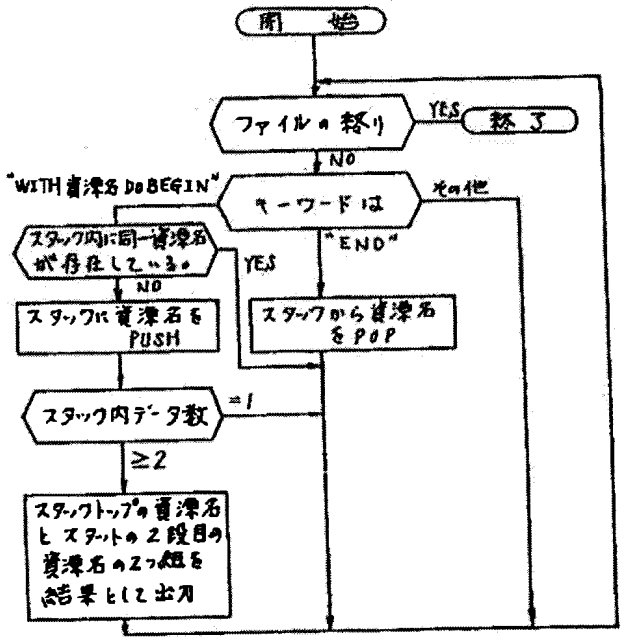
第１図



第２図



第３図



第4図

(a)
<出力結果>

(終点，始点)
(資源A，資源B)
(資源B，資源C)
(資源A，資源B)
(資源A，資源C)

(b)
<対応する有向グラフ>

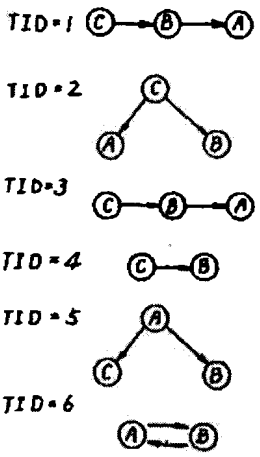
第5図

(a)

TID	始点	終点
1	B	A
1	C	B
2	C	B
2	C	A
3	C	B
3	B	A
4	C	B
5	A	C
5	A	B
6	A	B
6	B	A

10

(b)
<対応する有向グラフ>

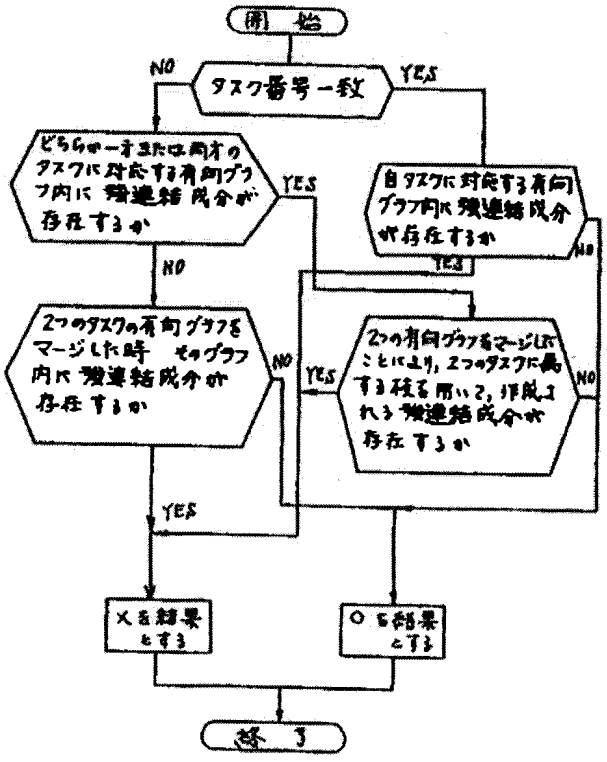


第6図

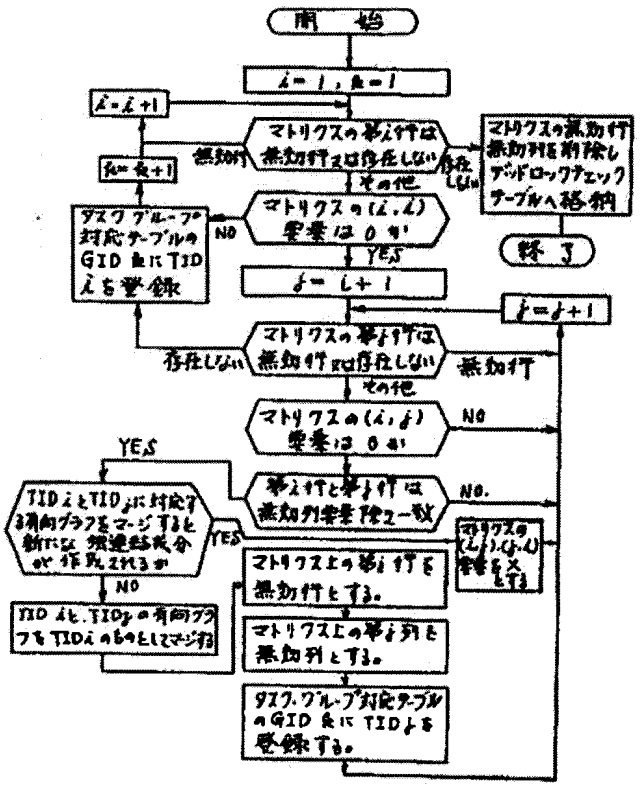
TID \ TID	1	2	3	4	5	6
1	○	○	○	○	×	×
2	○	○	○	○	×	×
3	○	○	○	○	×	×
4	○	○	○	○	○	○
5	×	×	×	○	○	×
6	×	×	×	○	×	×

14

第7図



第8図



第9図

(a)

GID \ GID	1	2	3	4
1	○	○	×	×
2	○	○	○	○
3	×	○	○	×
4	×	○	×	×

12

(b)

TID	GID
1	1
2	1
3	1
4	2
5	3
6	4

13

第10図